Supplemental Documentation

The following document contains figures, images, graphs, data that will supplement the understanding of our design and present evidence supporting its reliability and performance.

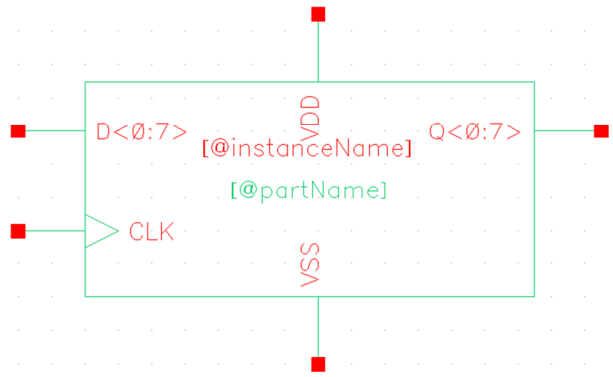
Robert Costanzo – rwc3bf

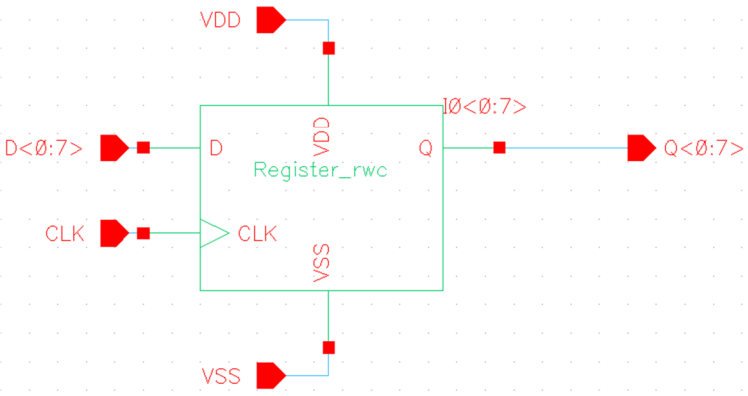
Austin Moran – apm4yw

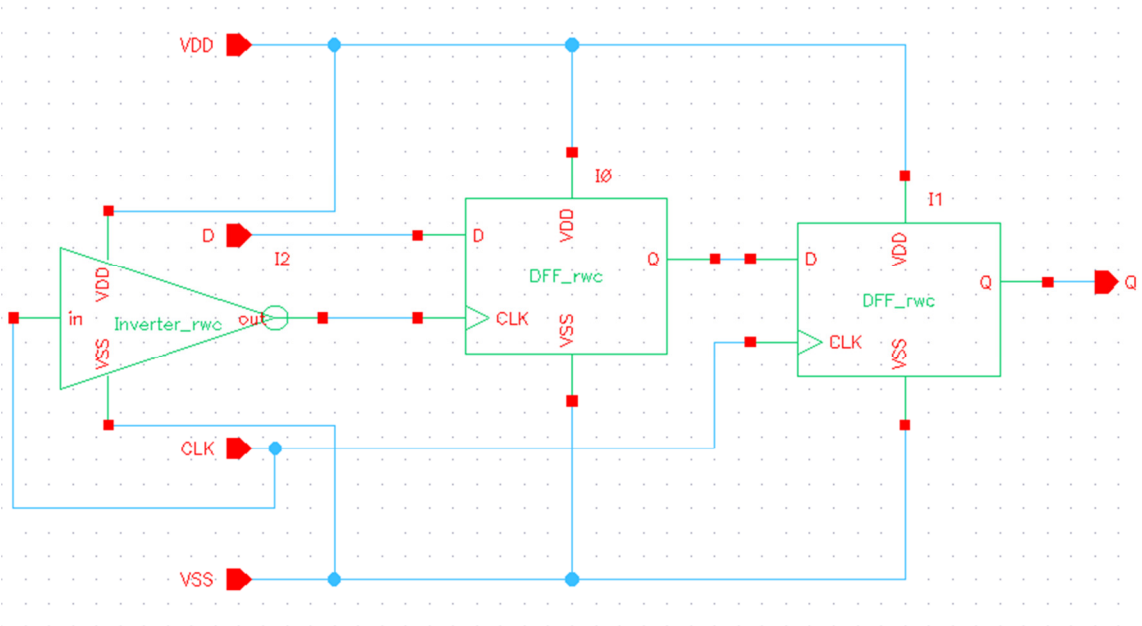
Hector Soto – hls6dc

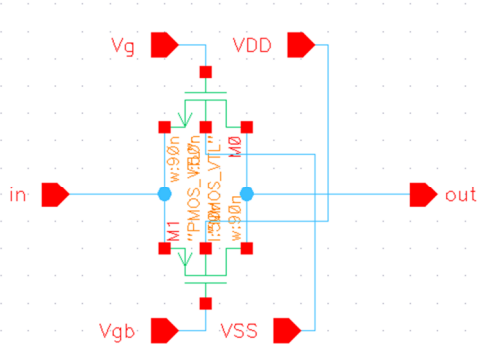
Final Product (Entire DSP ALU)

Registers

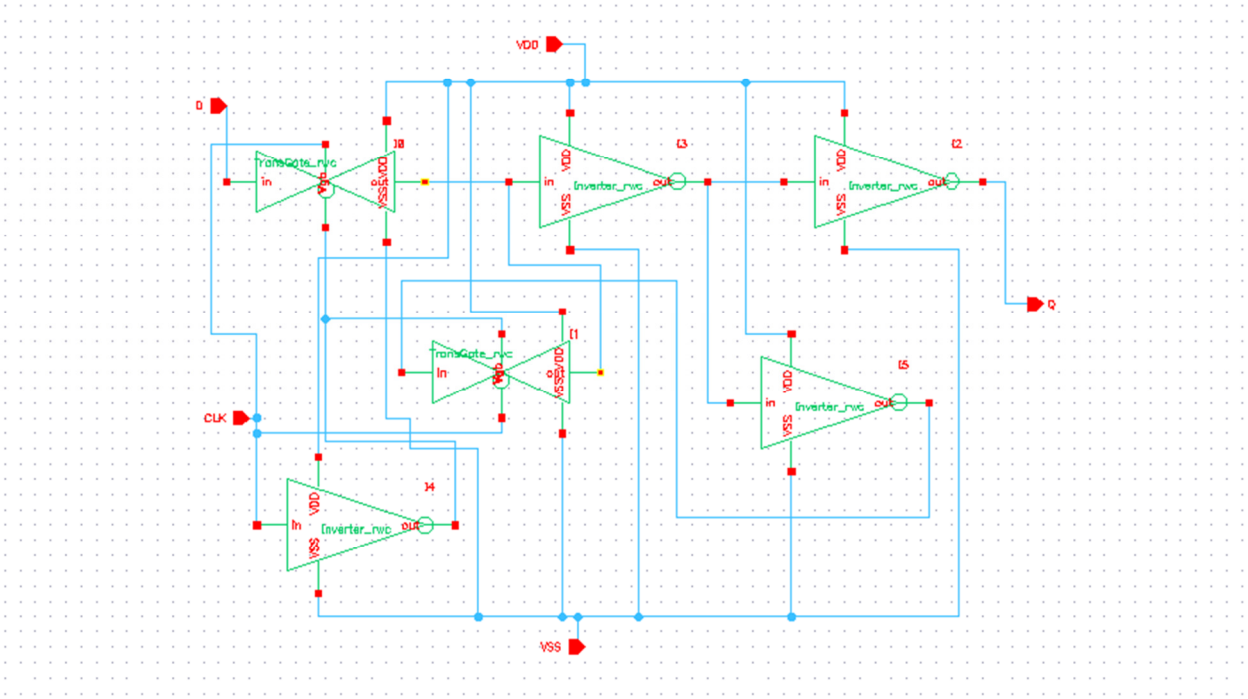




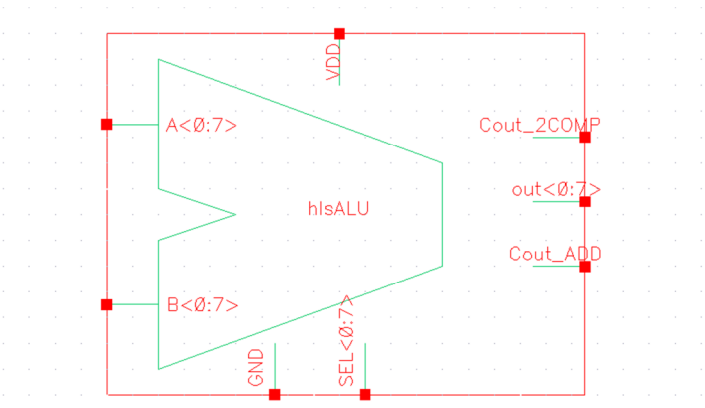


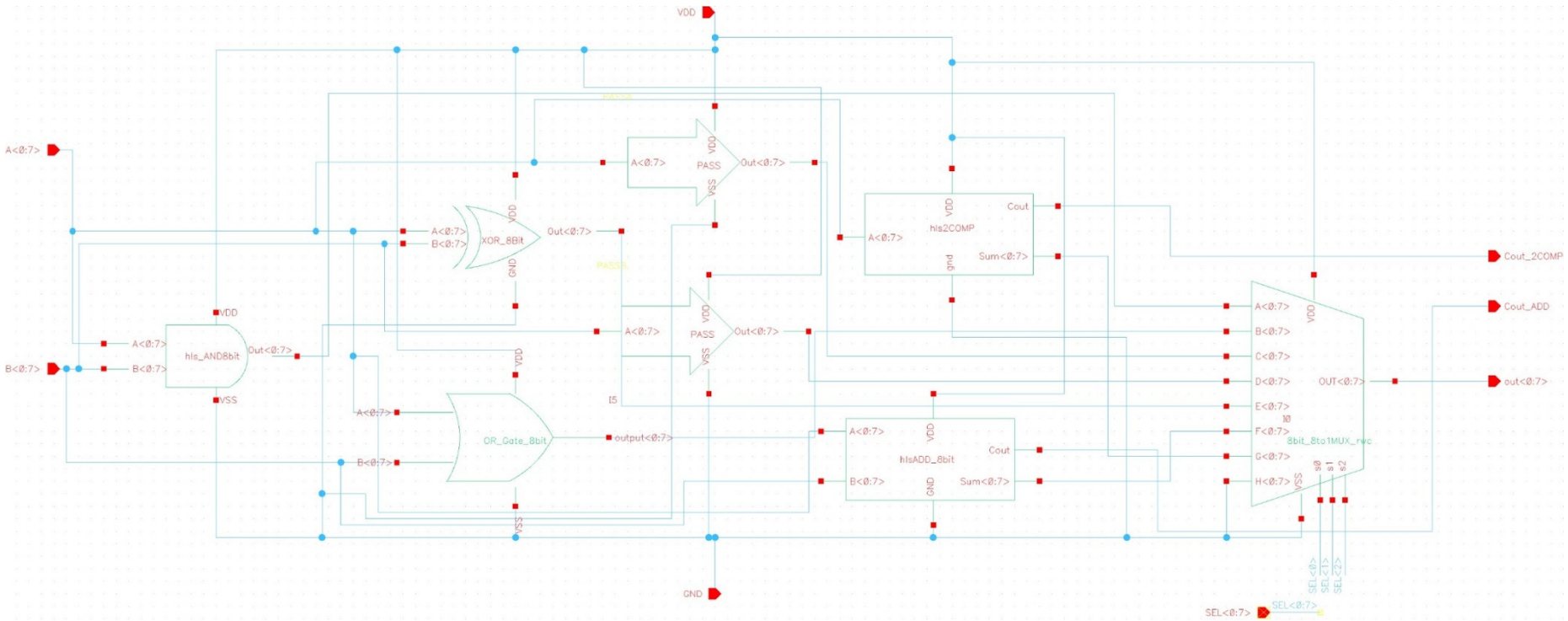


Transmission gate used in the Latch below

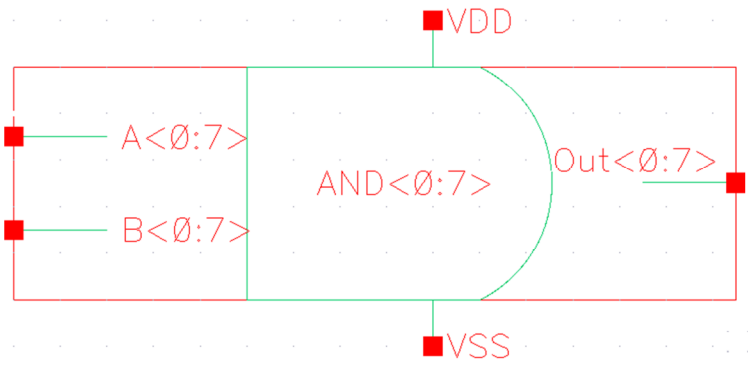


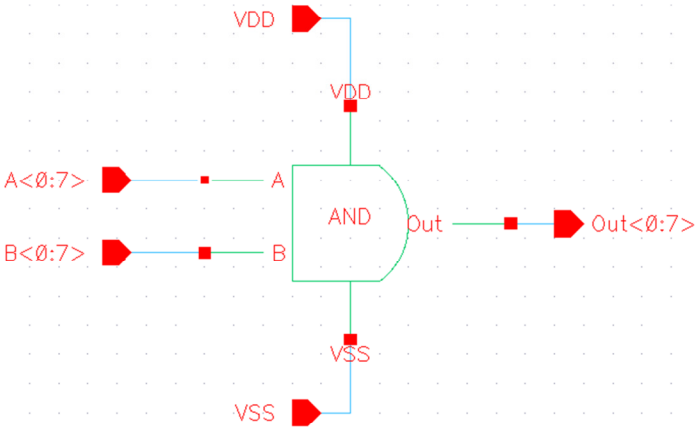
ALU

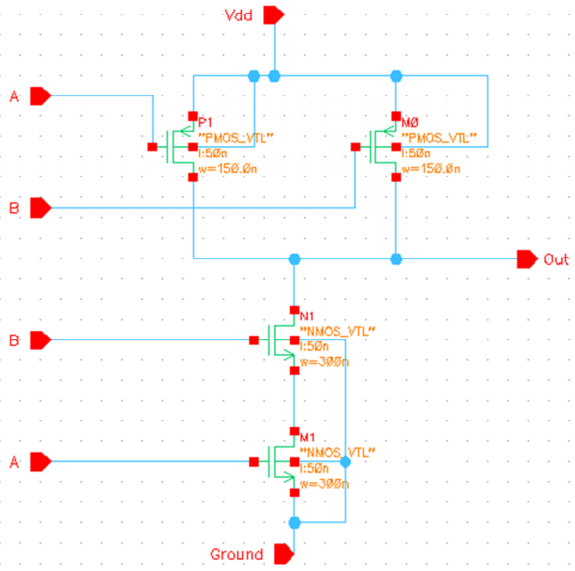



VDD
A<Ø:7>
B<Ø:7>
hls_AND8bit
Out<Ø:7>
VSS
XOR_8Bit
GND
PASS
OR_Gate_8bit
output<Ø:7>
hls2COMP
Cout
Sum<Ø:7>
hlsADD_8bit
C<Ø:7>
D<Ø:7>
E<Ø:7>
F<Ø:7>
G<Ø:7>
H<Ø:7>
OUT<Ø:7>
8bit_8to1MUX_rwc
s0
s1
s2
SEL<Ø:7>
Cout_2COMP
Cout_ADD
out<Ø:7>

AND

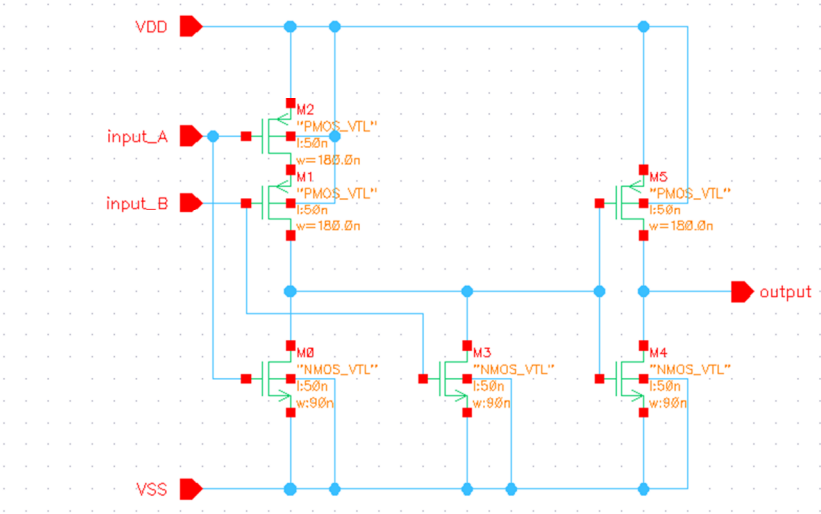




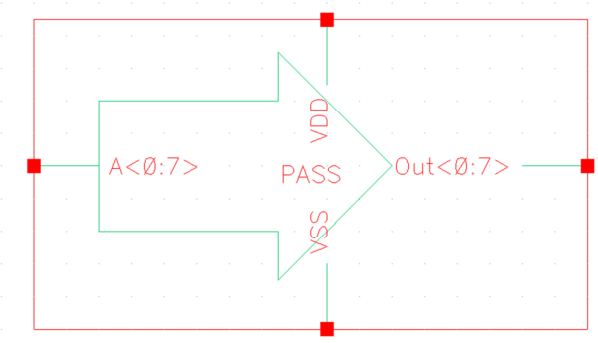


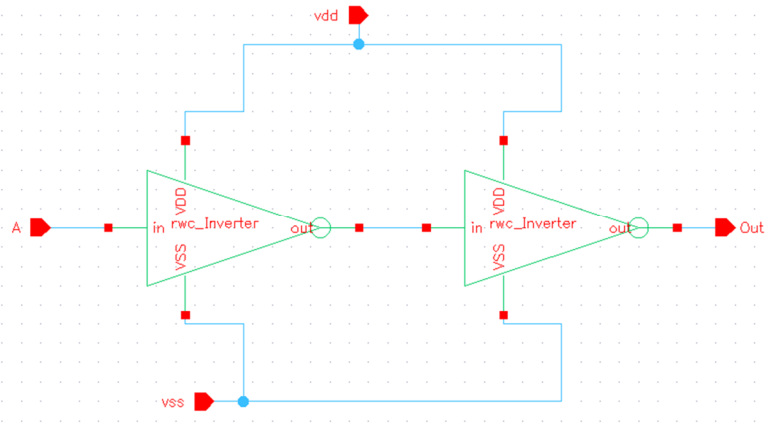
This shows the transistor schematic for a NAND gate, our AND gate was simply the final output.

OR

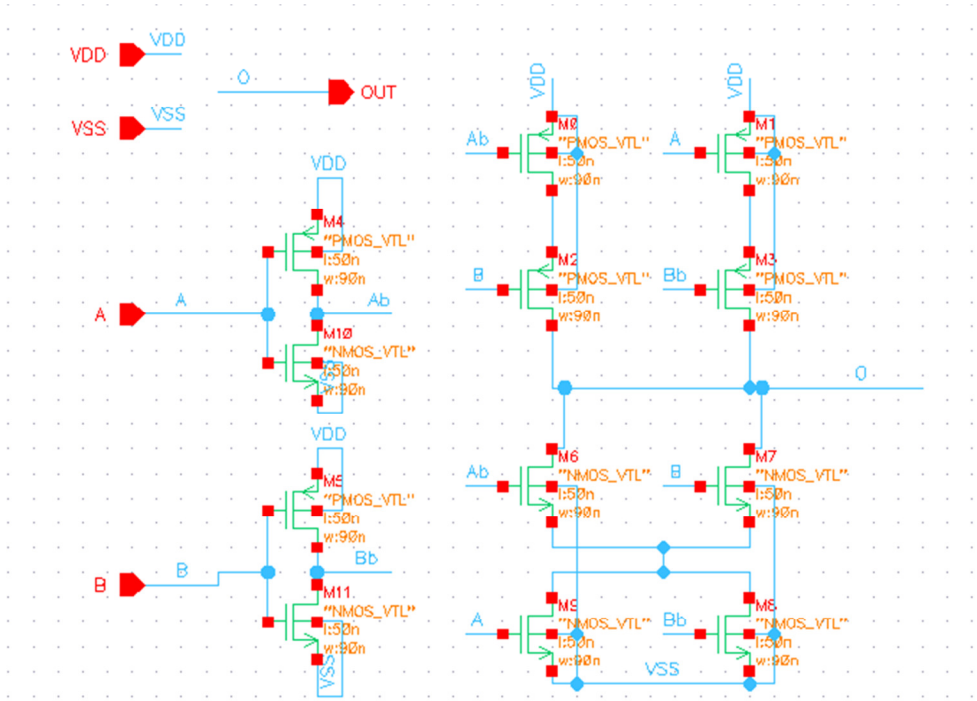


PASS-A / NO-OP (PASS-B)

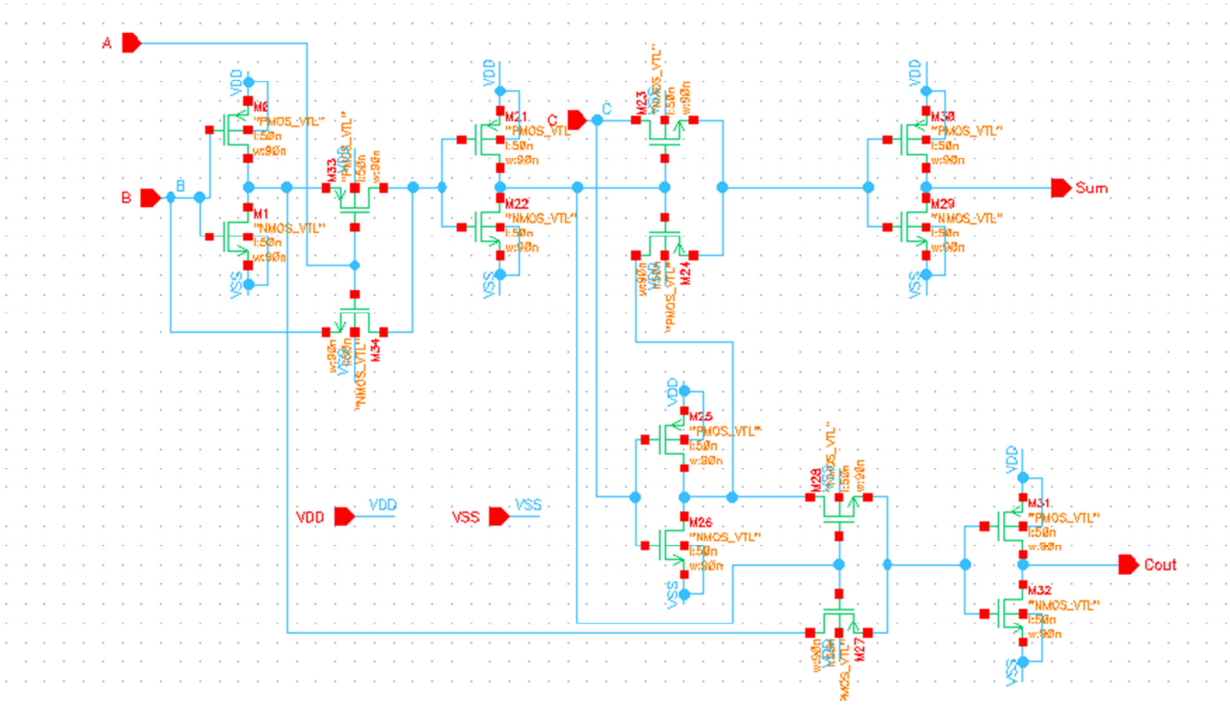




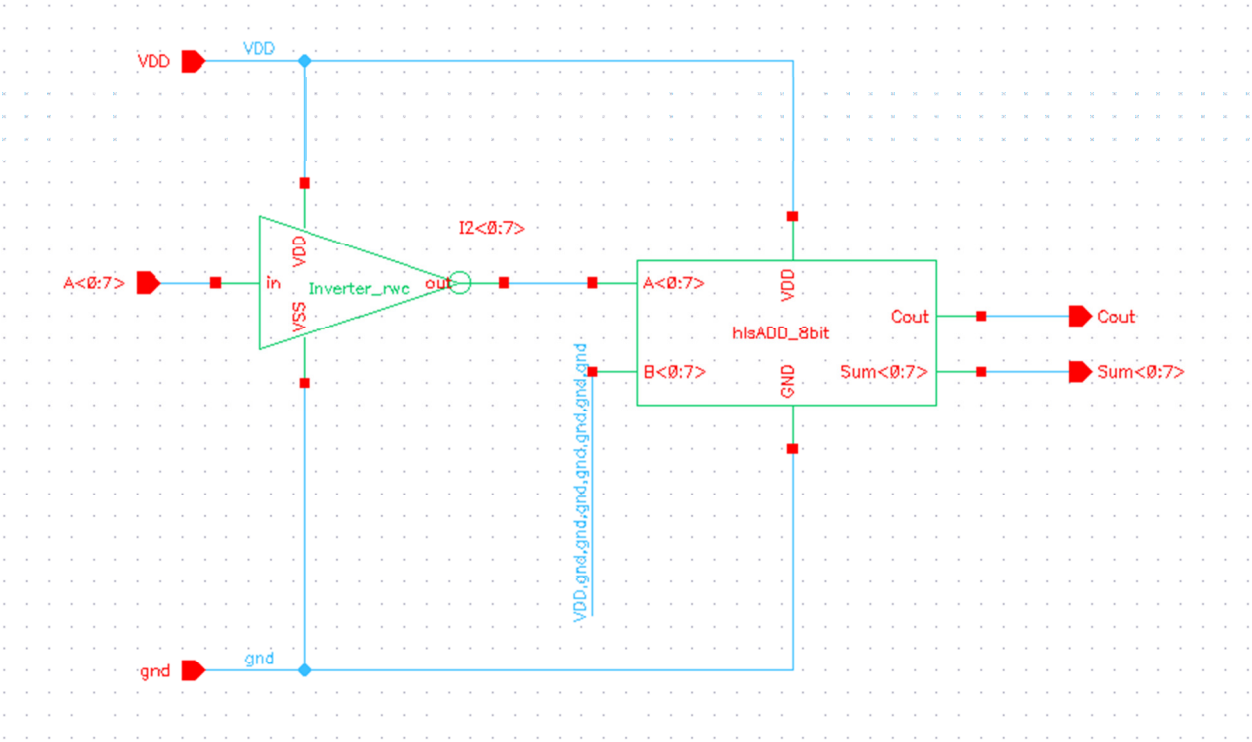
XOR (Arbitrary Function)



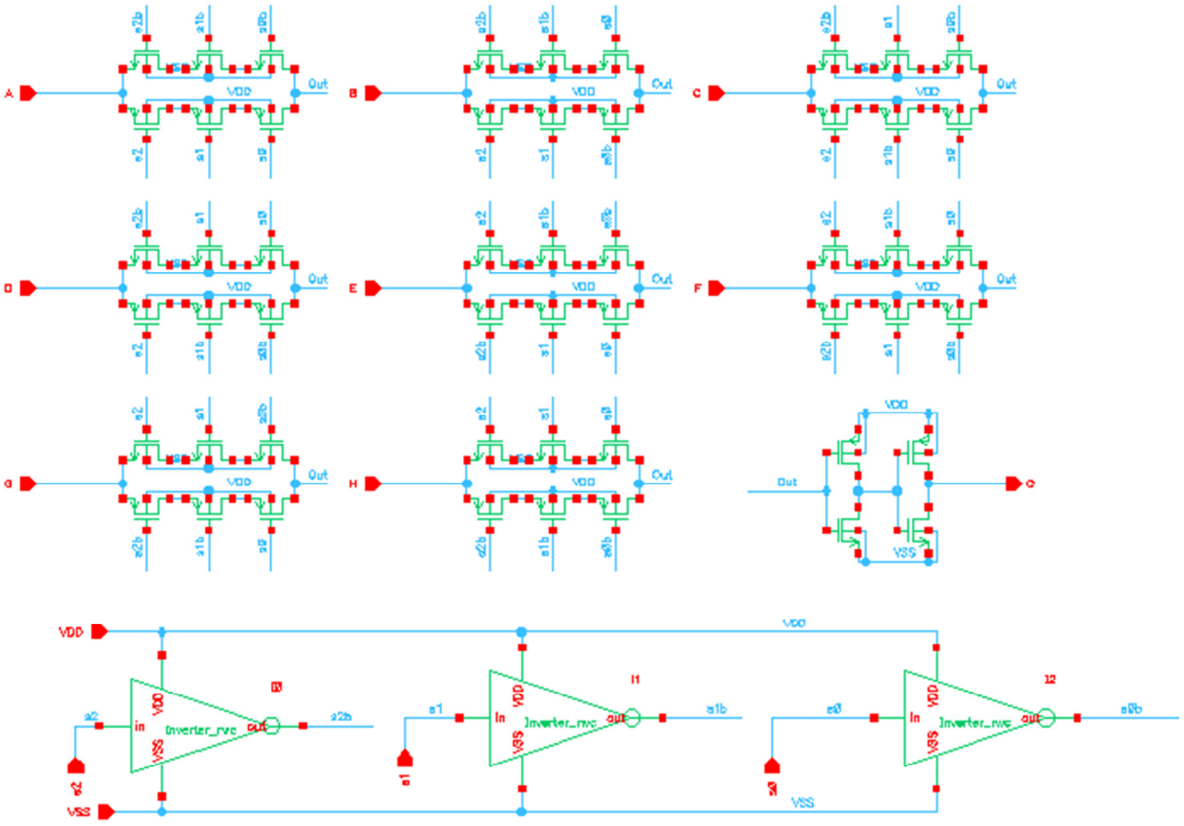
Adder



2’s COMP



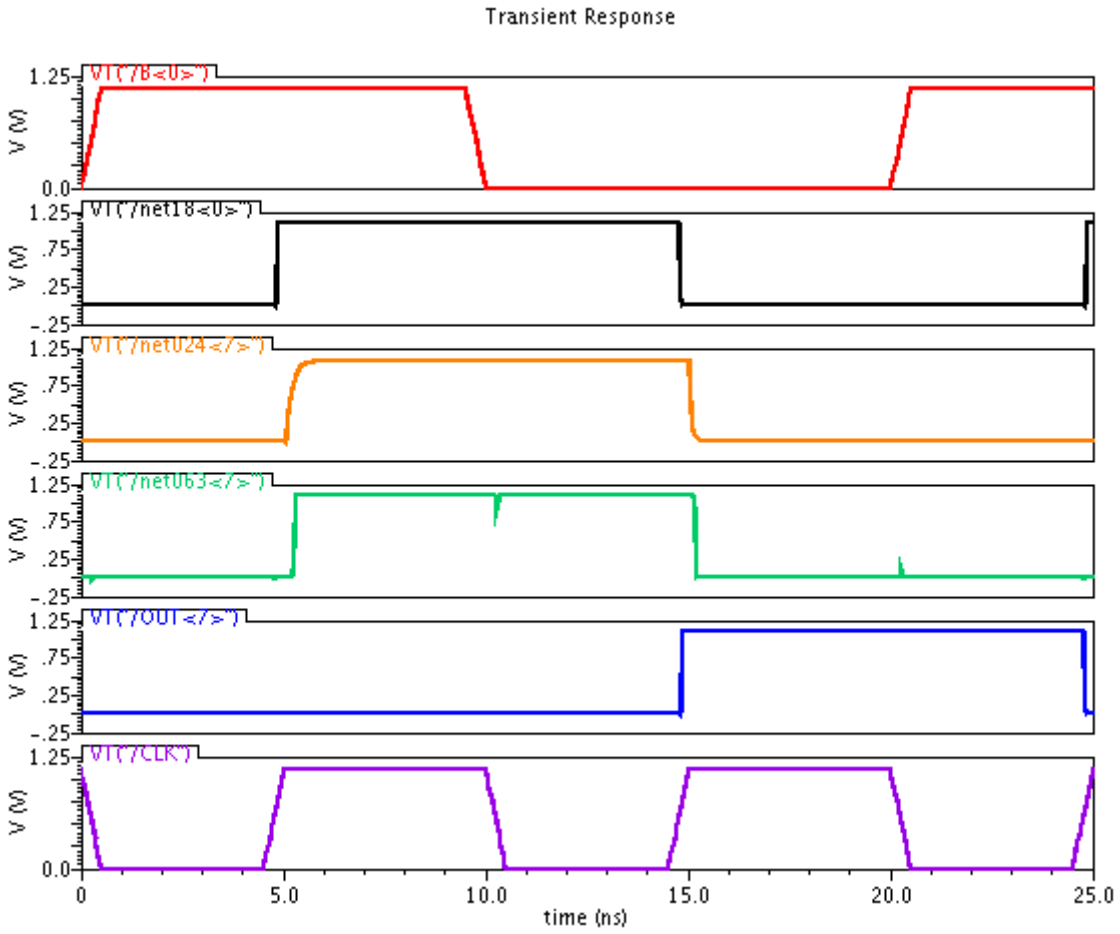
MUX



This may be hard to see, but each of the 8 inputs are transmitted through a series of 3 transmission gates. Each input (A through H) is transmitted if a unique set of s2, s1, and s0 is currently applied. The output (ie, the selected input) is then put through a double inverter to reduce the effect of the lost threshold voltages from the transmission gates.

Supplemental Tests

Worst Case Delay



Here, the worst case was when the adder received A= 01111111 and B transitioned to 00000001. This would cause the critical path of the carry ripple to propagate all the way through. Therefore, we only had to watch the delay from when B<0> to the A<7>. The plots above are:

Red: B<0> pulsing.

Black: The value of Q in the register that holds the value of B<0>.

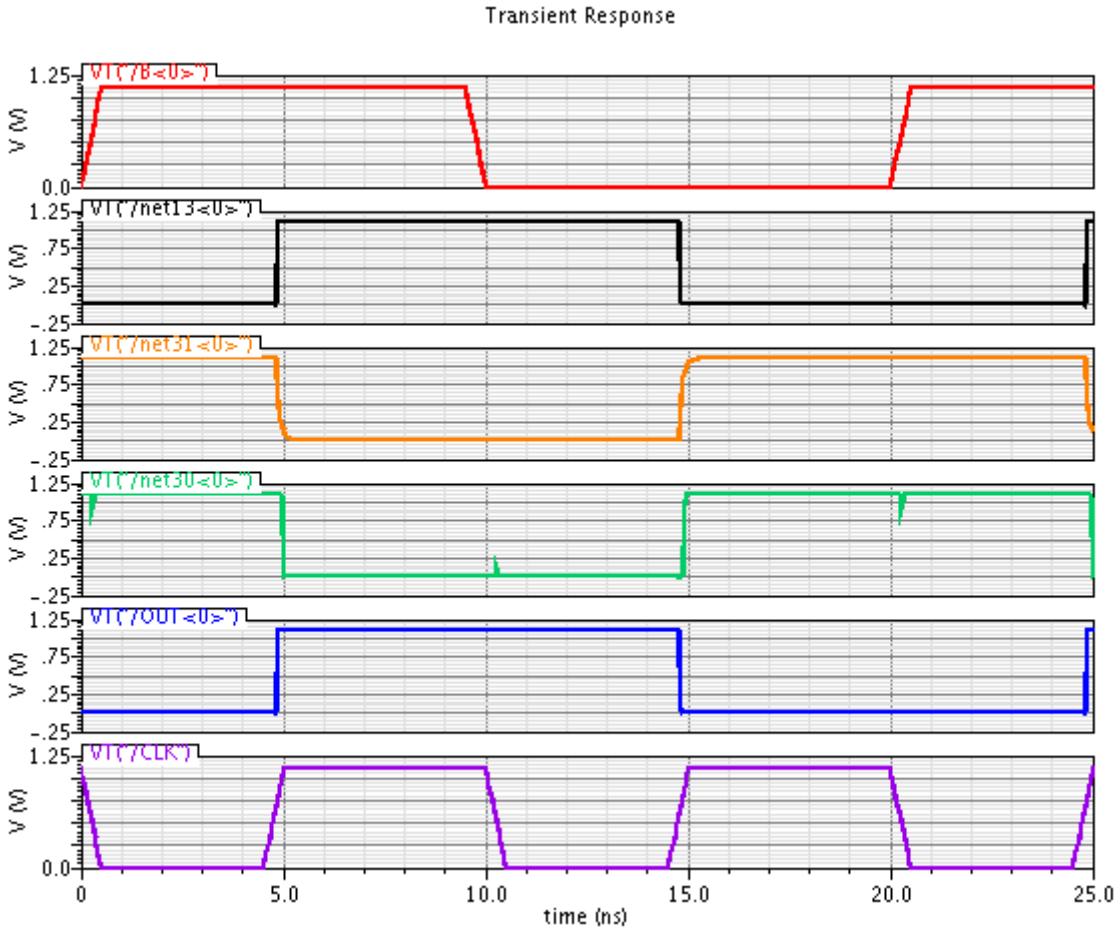
Orange: The Output<7> of the adder.

Green: The output of the MUX<7>

Blue: The value of Q in the register that holds the output of the ALU<7>

Purple: CLK

Worst Case Delay of Arbitrary Function



Here, we kept A as 11111111 and B was pulsing between 00000000 and 11111111. We only graphed bit 0 of each as they were all identical and thus representative. The graph above is as follows:

Red: B<0> pulsing.

Black: The value of Q in the register that holds the value for B<0> and inputs it to the XOR.

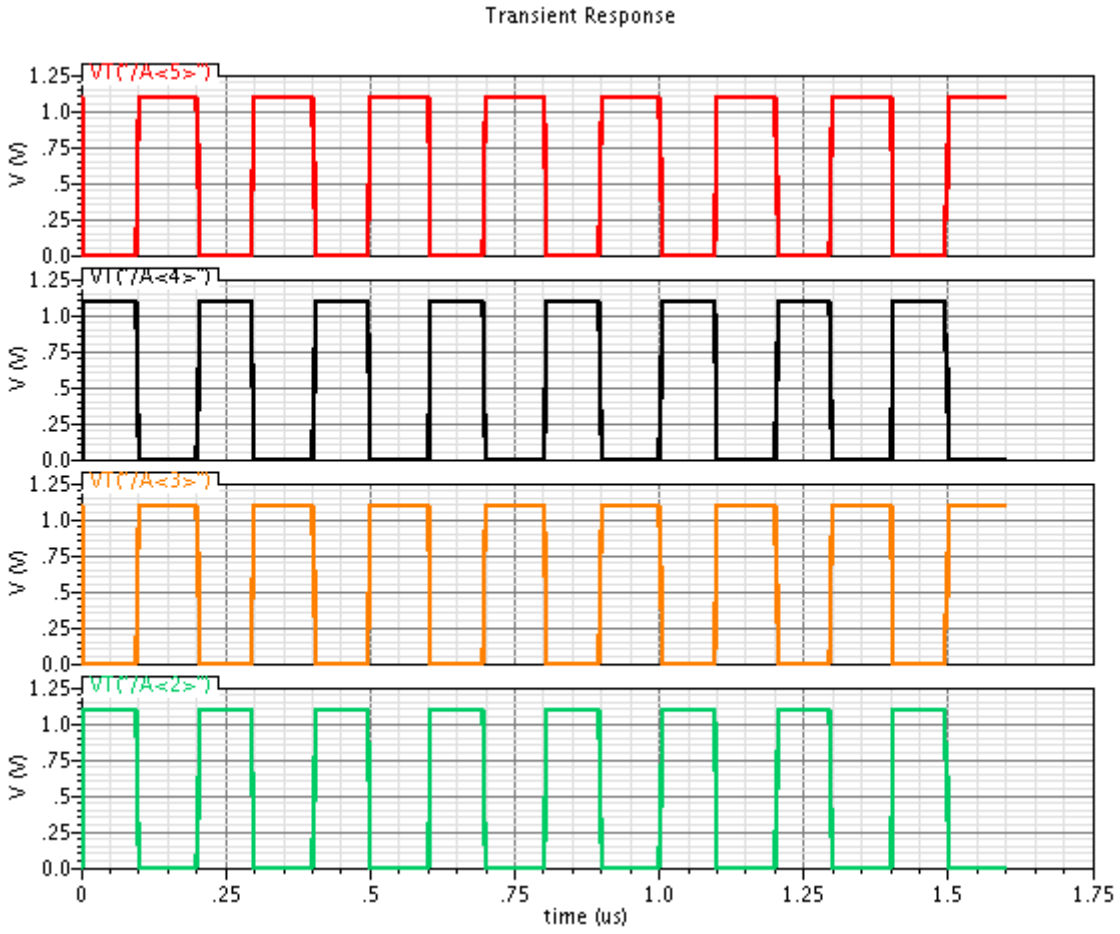
Orange: The output of the XOR<0>.

Green: The value output through the MUX<0>.

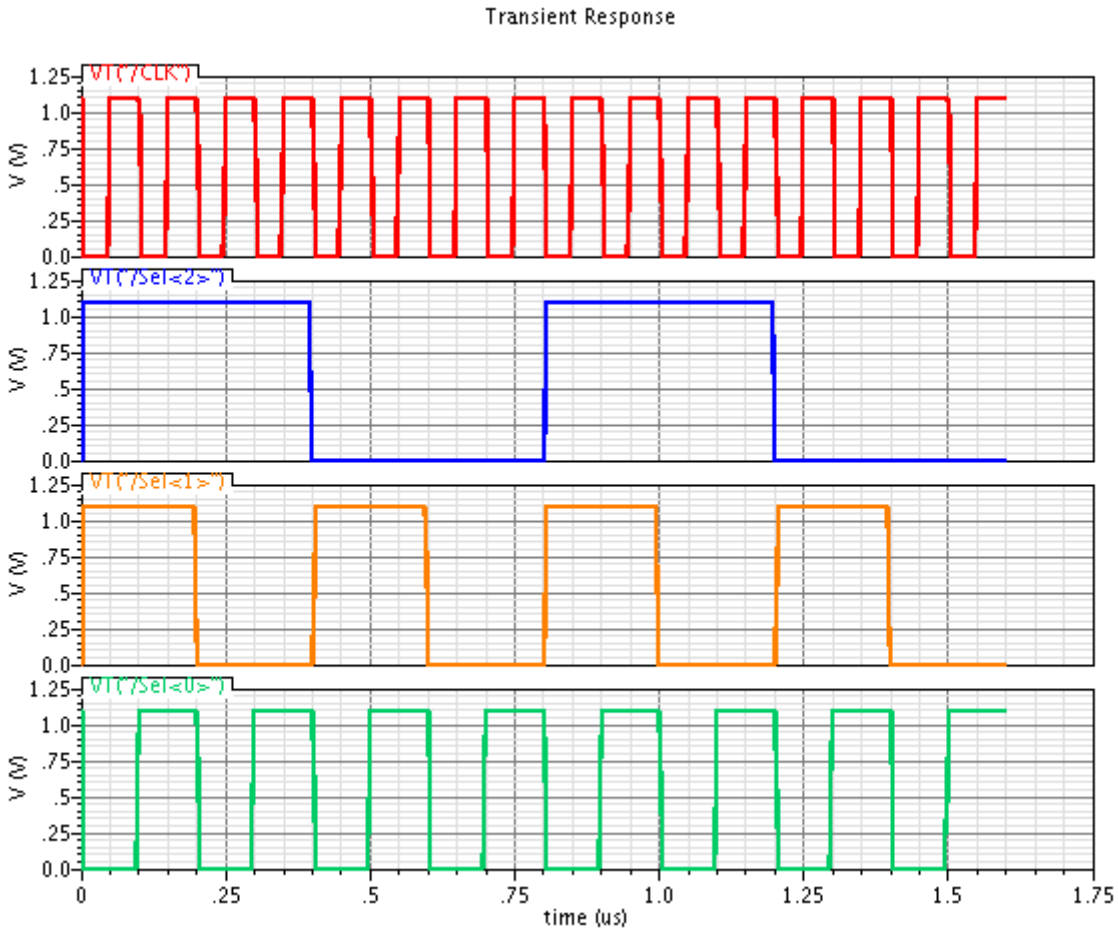
Blue: The value of the Q in the register that latches the output value of the ALU<0>.

Purple: CLK

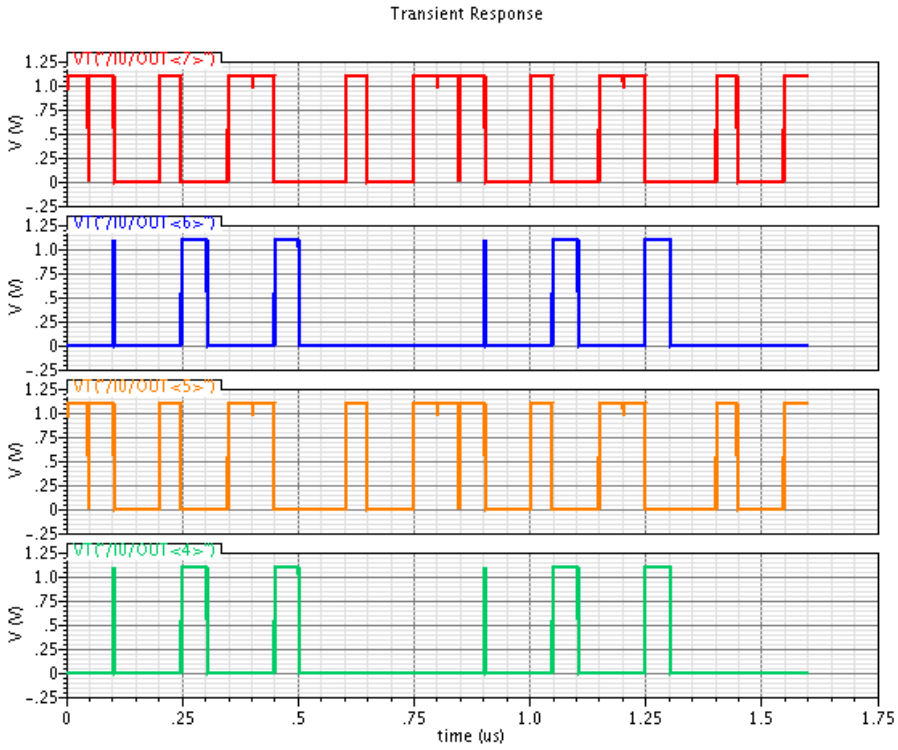
Energy Calculations

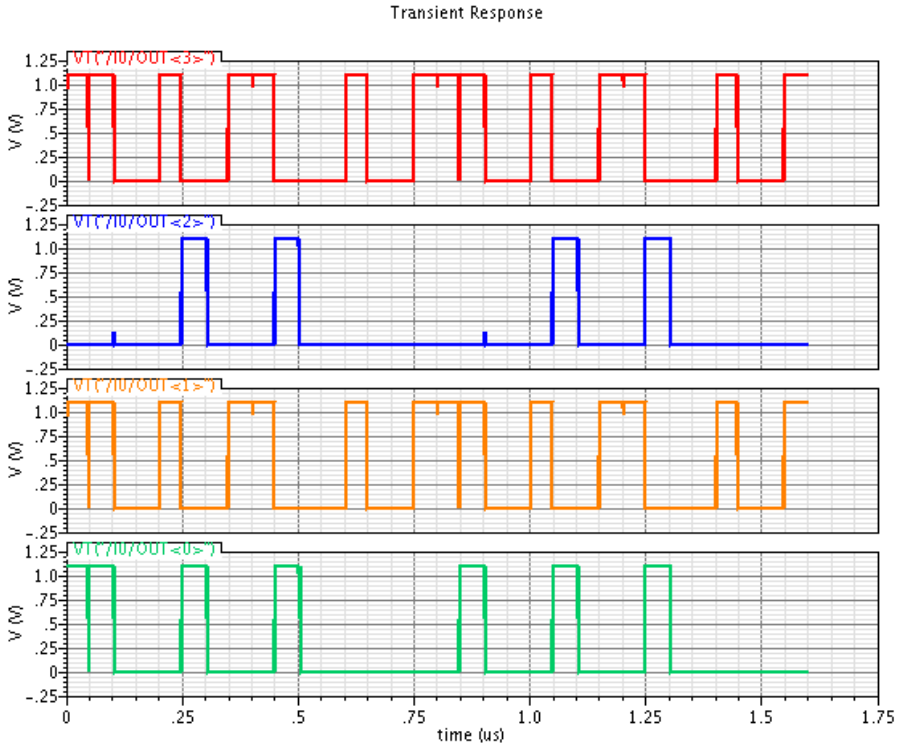


This is the graph of the A<2:5> pulsing from 0101 and 1010.



This is the graph of CLK (red), Sel<2> (blue), Sel<1> (orange), and Sel<0> (green).

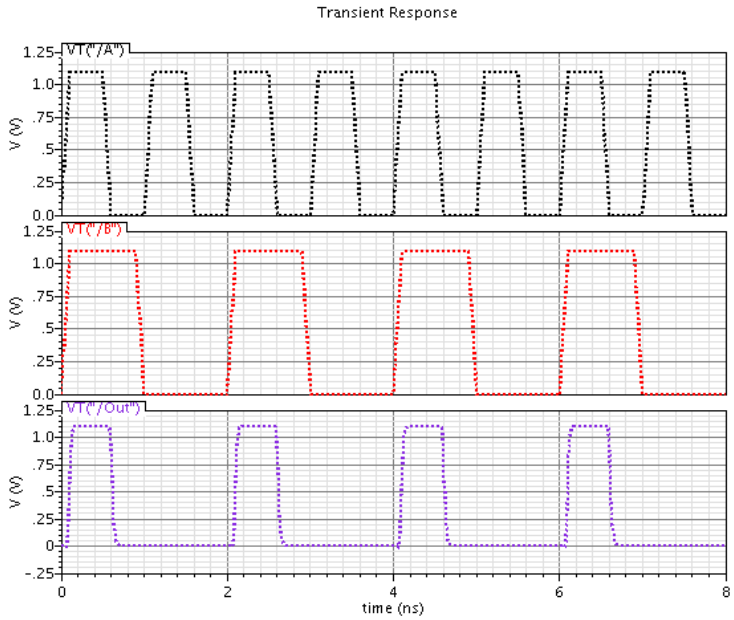




The above 8 plots are, in descending order, DSPOUT<7> through DSPOUT<0>.

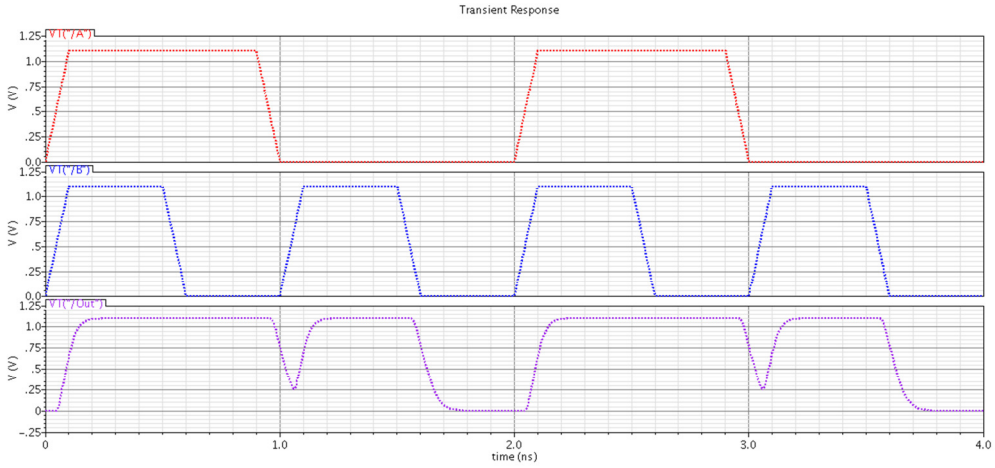
Functionality Tests

AND



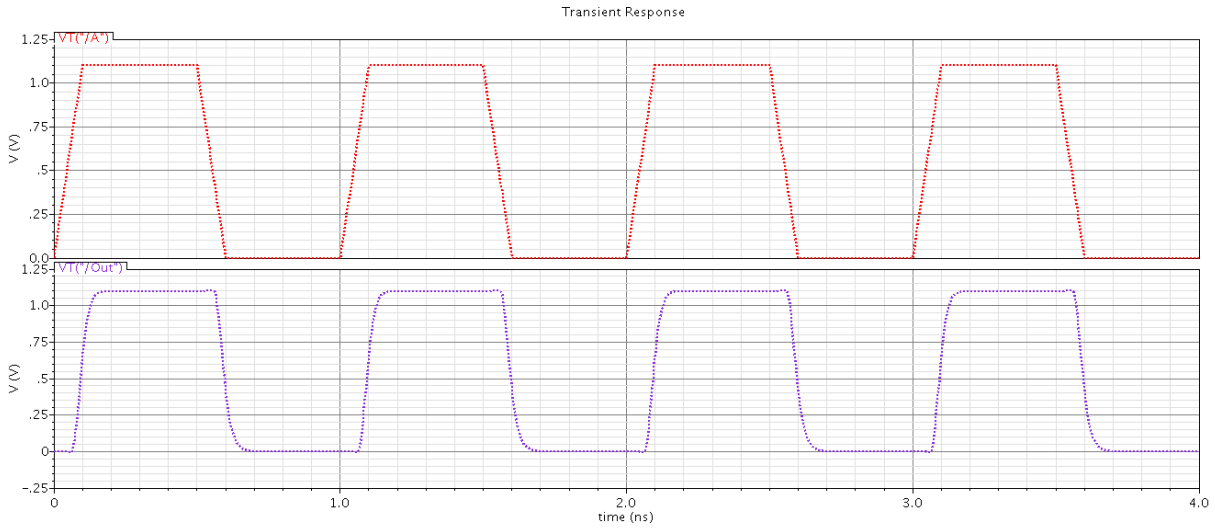
The black graph represents the alternating signal of one bit in the “A” signal entering the AND and the red graph represents the alternating signal of one bit of the “B” signal entering the AND. The purple graph represents the output produced by the combination of the “A” and “B” input signals. All bits showed similar responses and behaved accordingly.

OR



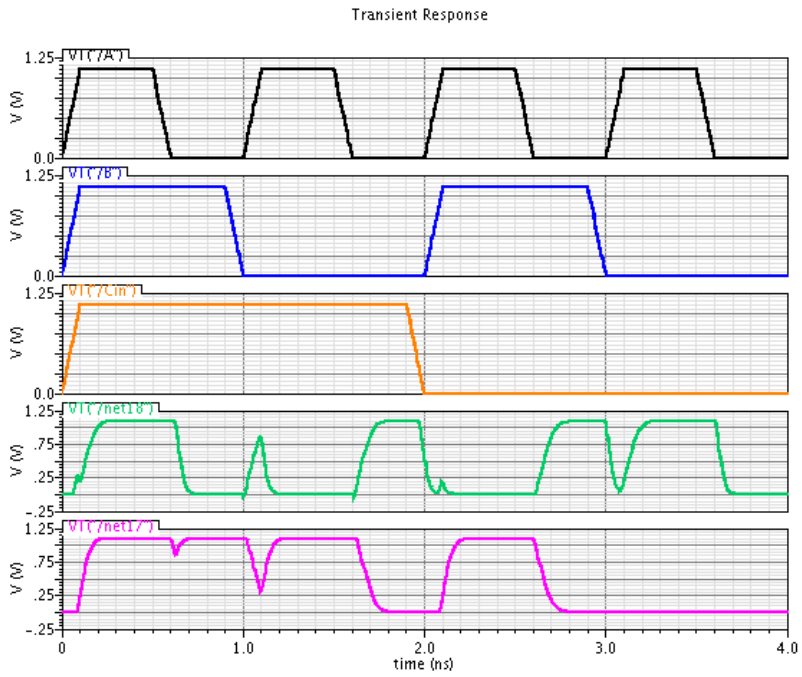
Red is input A, Blue is input B, and Purple is the output. The graph shows that the gate is functioning properly. The decrease in the output is due to both inputs being below 0.5 VDD. All input combinations (4) were tested by alternating input B at half the rate of input A, this cycles through all of the states quite nicely.

PASS (NO-OP)



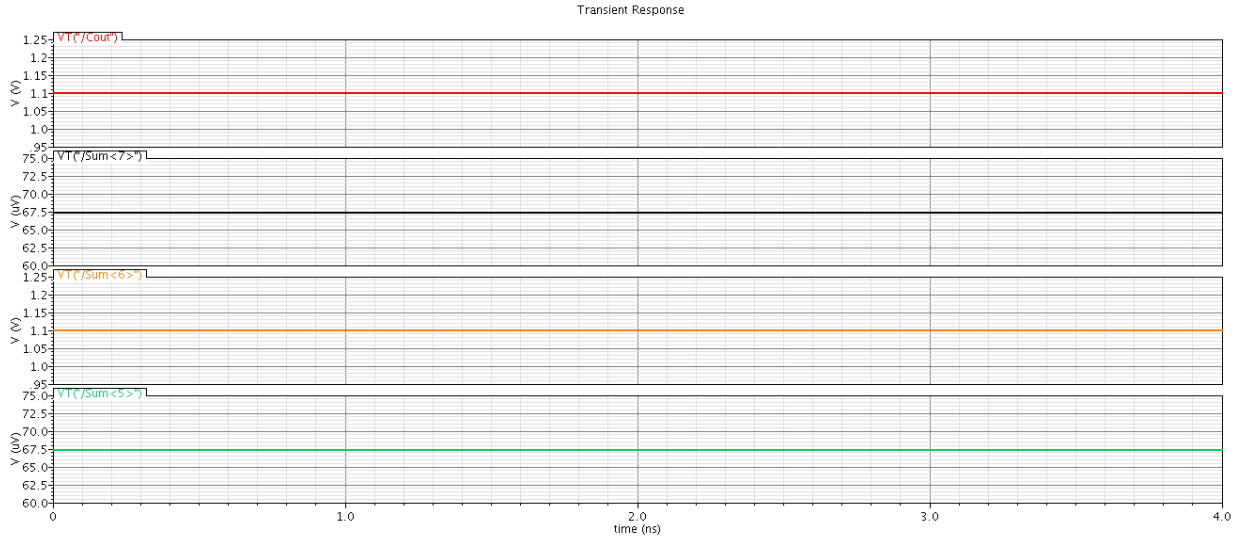
The transient graph above shows that the signal remains unaltered for each bit, though only one bit is shown here, and there is only a slight delay in the rise and fall times of the output signal compared to the input.

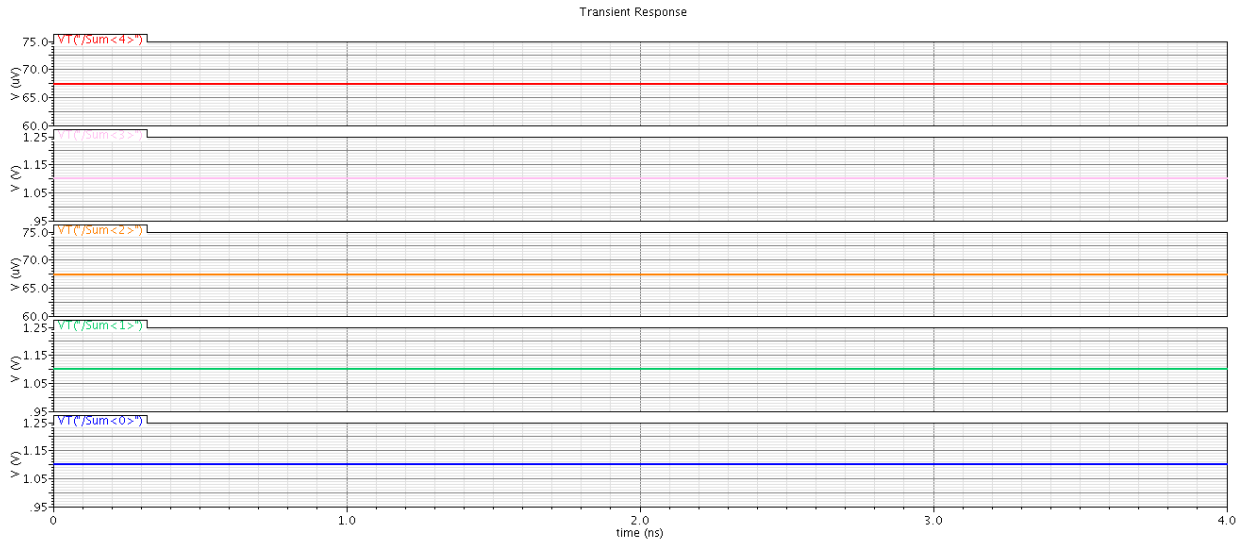
ADDER



Here we see the first 3 transient curves are the pulses used as inputs to A, B, and Cin respectively. The green graph is the Sum output and the purple graph is the Cout. Clearly, the two outputs act appropriately and the glitches are created by the pulses transitioning at the same time, causing the system to behave irregularly until the inputs pass the logic transition thresholds.

Then, we tested a few arbitrary cases at the 8-bit level to see if the interconnection of all the appropriately working 1-bit adders also worked correctly. One of the cases we used and are showing in this design review is the addition of the two 8-bit binaries - 01110101 and 11010110. The resulting output is 101001011 (ie. the 8-bit 01001011 and a carry out of 1). This is what we obtained during testing.

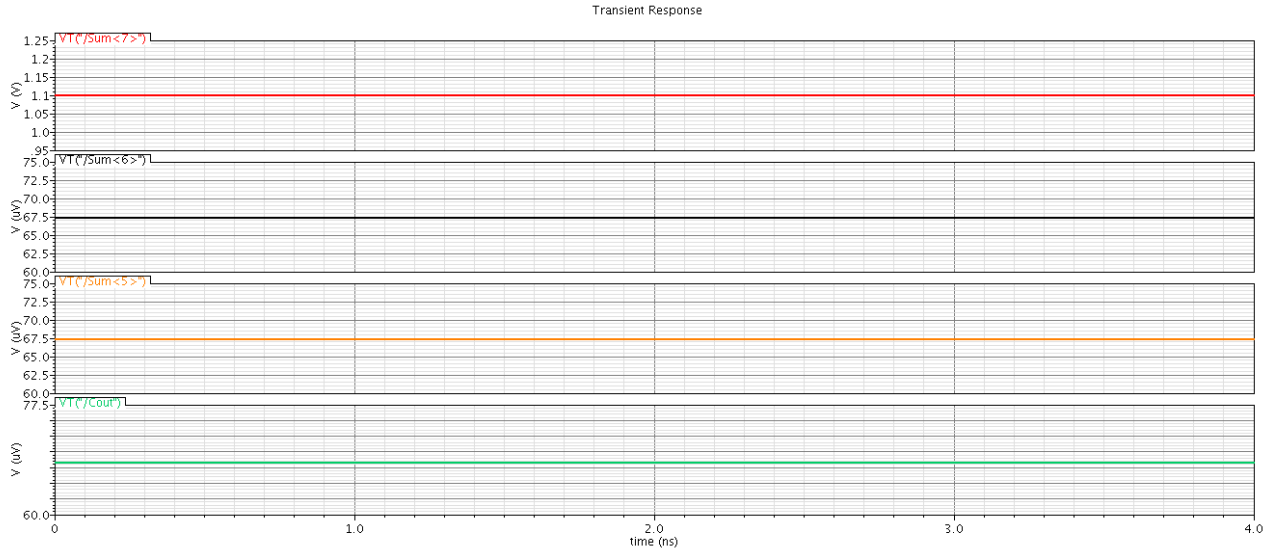


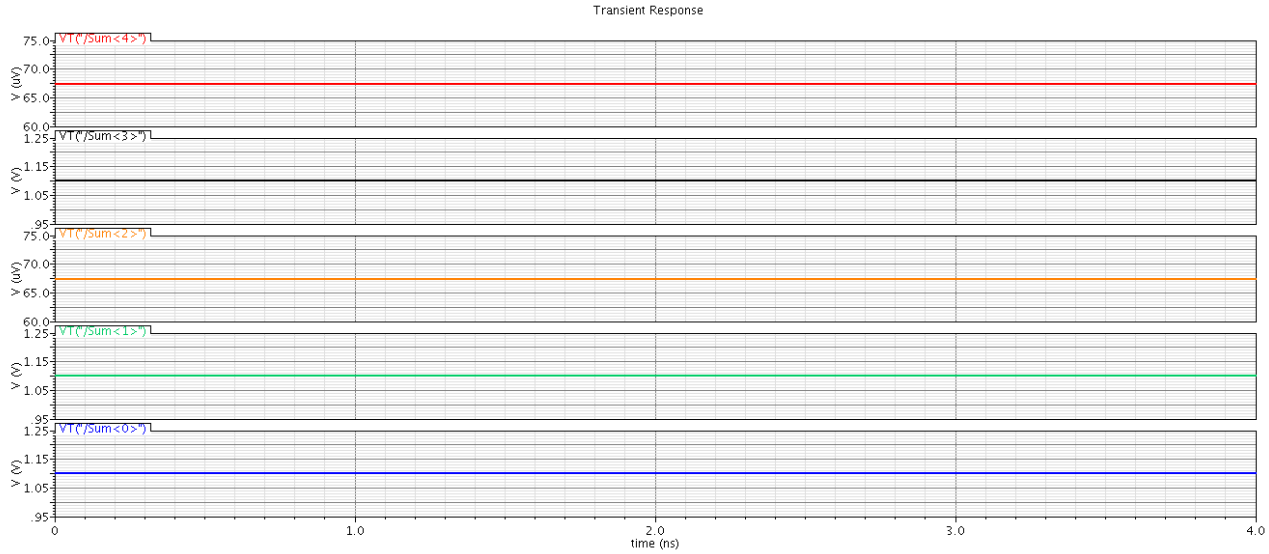


The two graphs shown above are of the 8-bit output and the final Cout. The very first plot is Cout (red) at a high level. The next (black) is Bit 7 at a low value. The next (yellow) is Bit 6 at a high. The next (green) is Bit 5 at a low. On the second graph, Bit 4 (red) is at a low value. Bit 3 (pink) is at a high value. Bit 2 (yellow) is at a low value. Bit 1 (green) is at a high value. Lastly, bit 0 (blue) is at a high value. Therefore, as desired, the output is 01001011 with a Cout of 1. Thus, we conclude that our 8-bit adder is working as desired.

2’s COMP

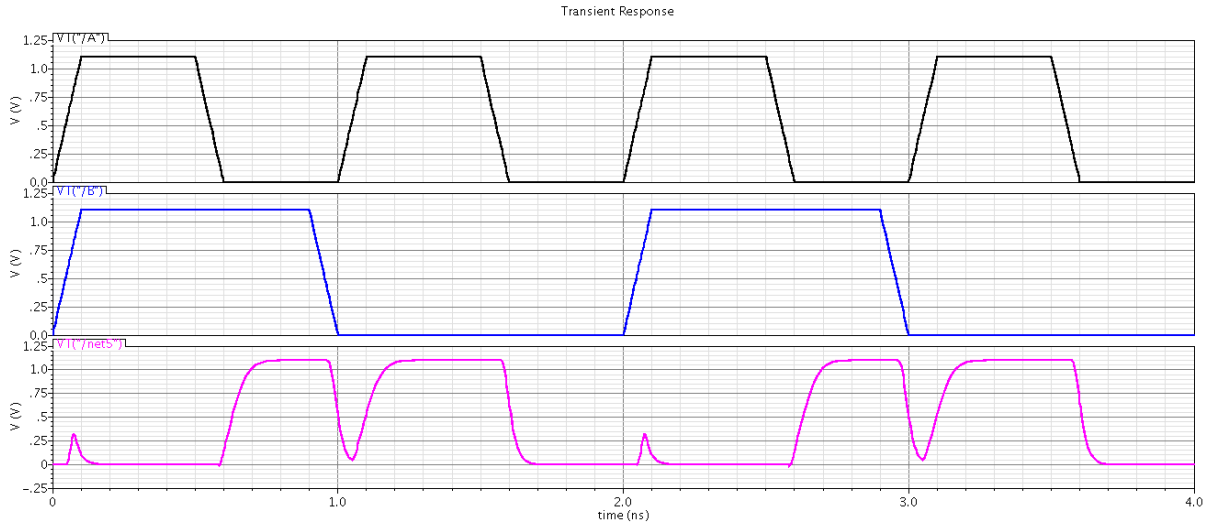
We cannibalized our 8-bit adder to create the 2’s complement. To obtain the 2’s complement of an 8-bit binary number, we inverted each bit of the input and then added 1 (or 00000001) to it. Therefore, we tied the inverted input into A of the adder, and then hardwired the 00000001 using an array of VDD and ground into B of the adder. The resulting output works as desired. We tested values such as 01110101 (which in 2’s complement is 10001011). The resulting response is pictured below:





The result of the Cout was low. This is green plot on the first graph above. The Bit 7 (red) was high. The Bit 6 (black) was low. The Bit 5 (yellow) was low. The Bit 4 (red of the second graph) was low. The Bit 3 (black) was high. The Bit 2 (yellow) was low. The Bit 1 (green) was high. Lastly, the Bit 0 (blue) was high. This means our output was equivalent to 10001011, exactly as desired. Therefore, we conclude that our 2’s complement works as desired.

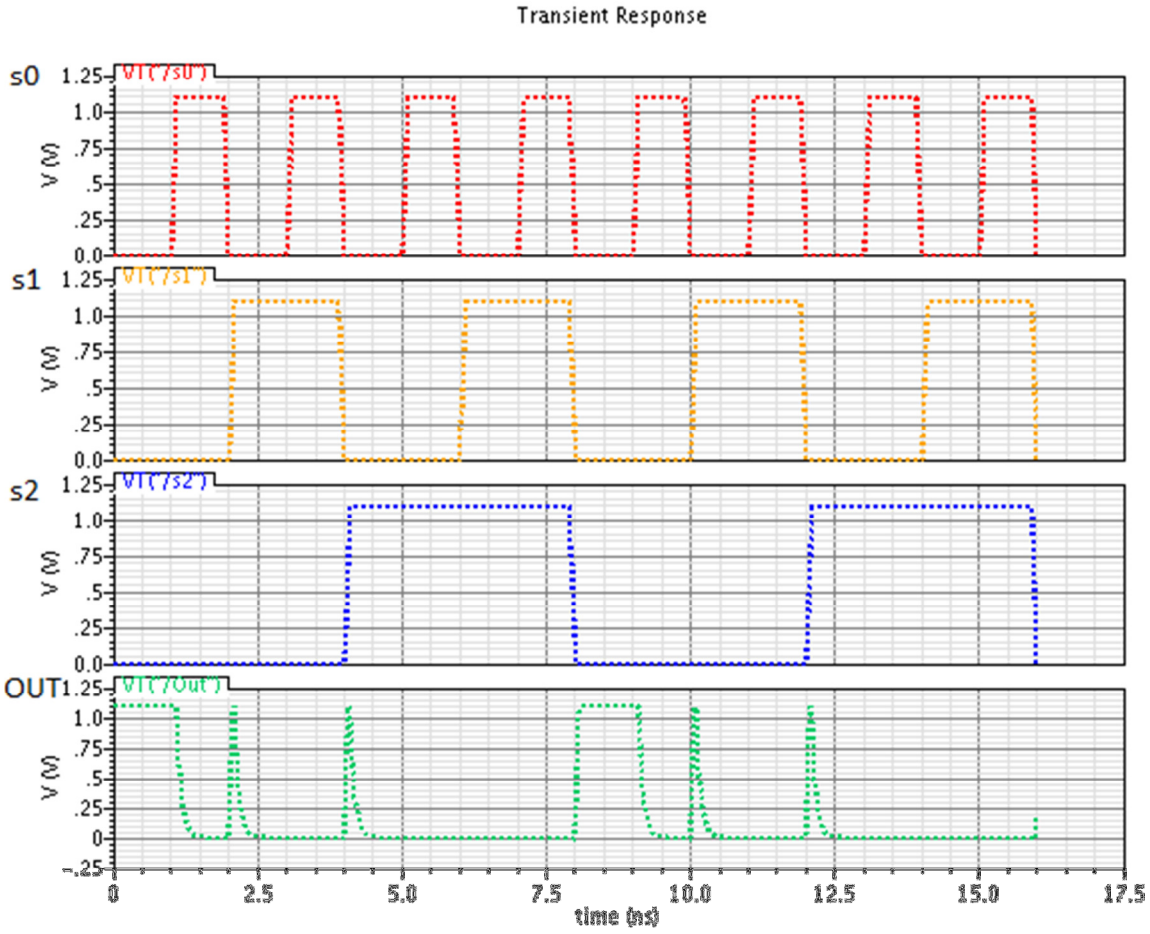
XOR



Black is input A, Blue is input B, and Purple is the XOR output. As we can see, our XOR gate works as desired

MUX

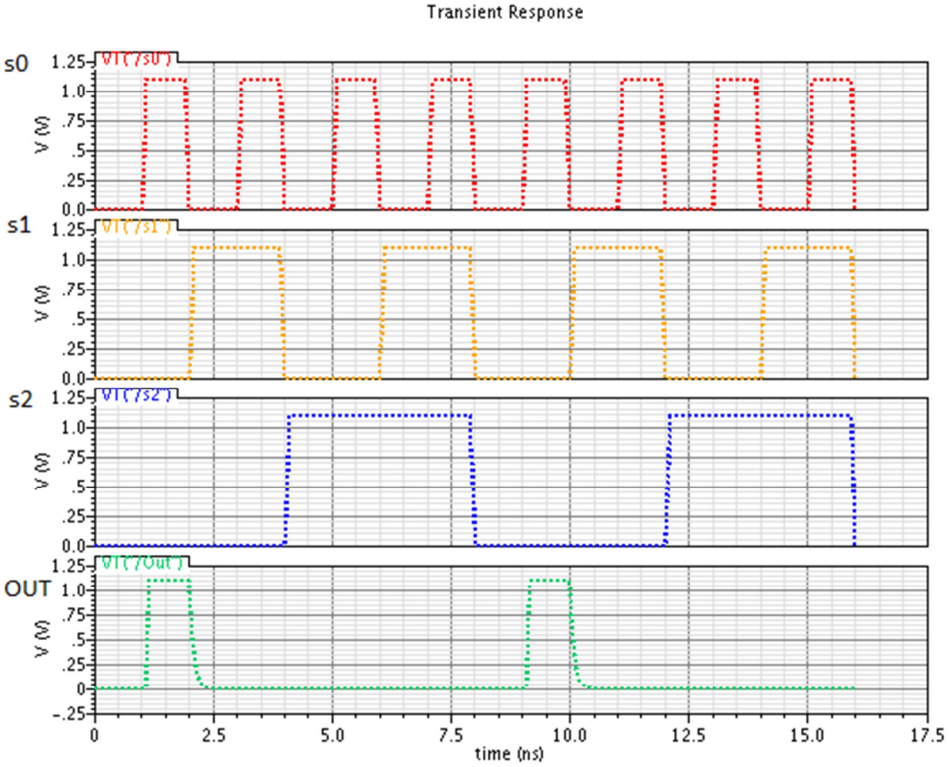
Graph of output when A is tied to high voltage : s2s1s0 = 000 yields high output as desired



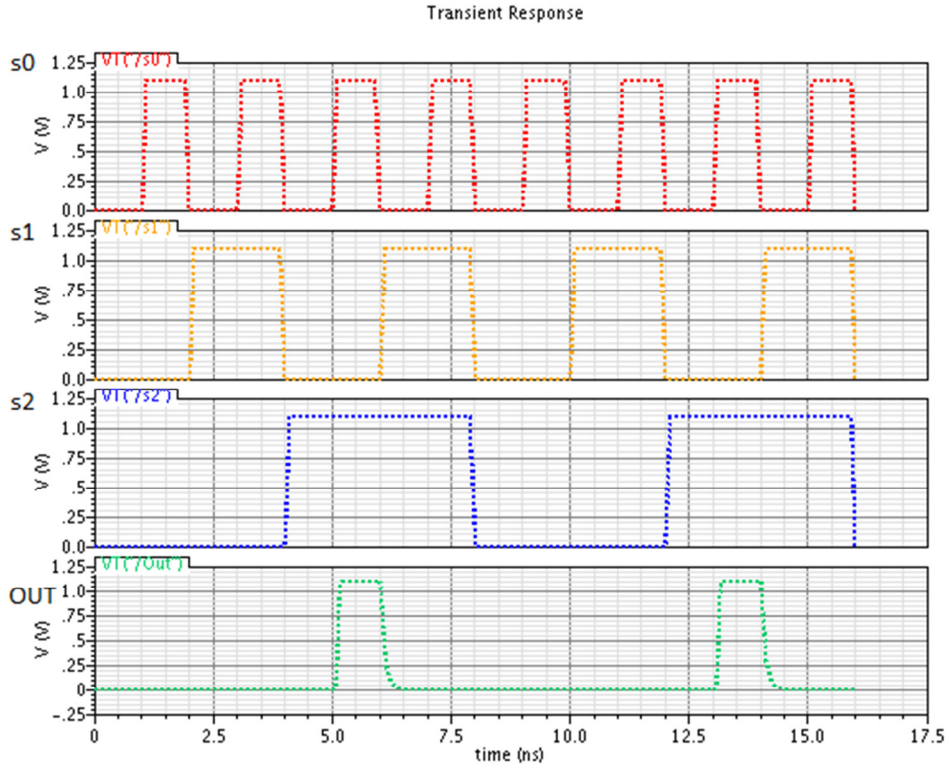
s0 – red, s1 – orange, s2 – blue, OUT – green

This graph is when only A was tied to a high voltage. This means that the only time the output should go high is when the selector lines s2s1s0 are logically 000 (or all low voltages). From the graph, we can see that this is indeed what happens. The output (green) only goes high when the selector lines are all low. However, we did find some glitches that are evident in the graph. During certain cases, the selector lines are transitioning, which causes the output to think they are all low and start outputting a high pulse. The rest of the cases (more shown below) seem to work fine without this glitch.

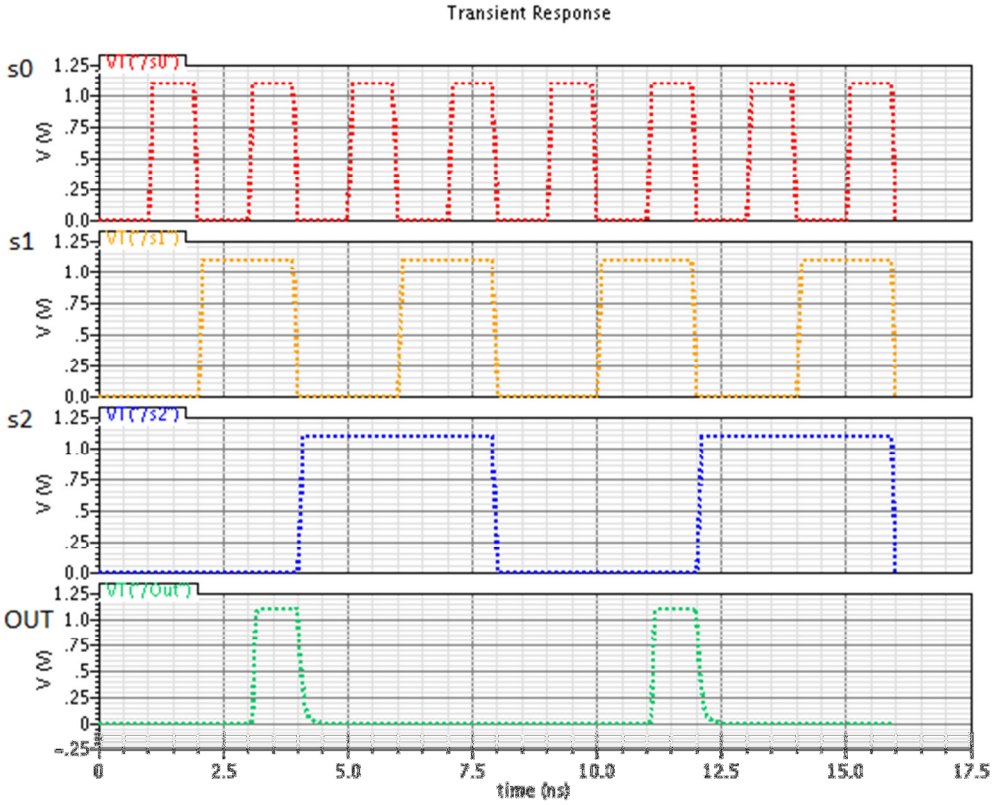
Graph of output when B is tied to high voltage : s2s1s0 = 001 yields high output as desired



Graph of output when F is tied to high voltage: s2s1s0 = 101 yields high output as desired

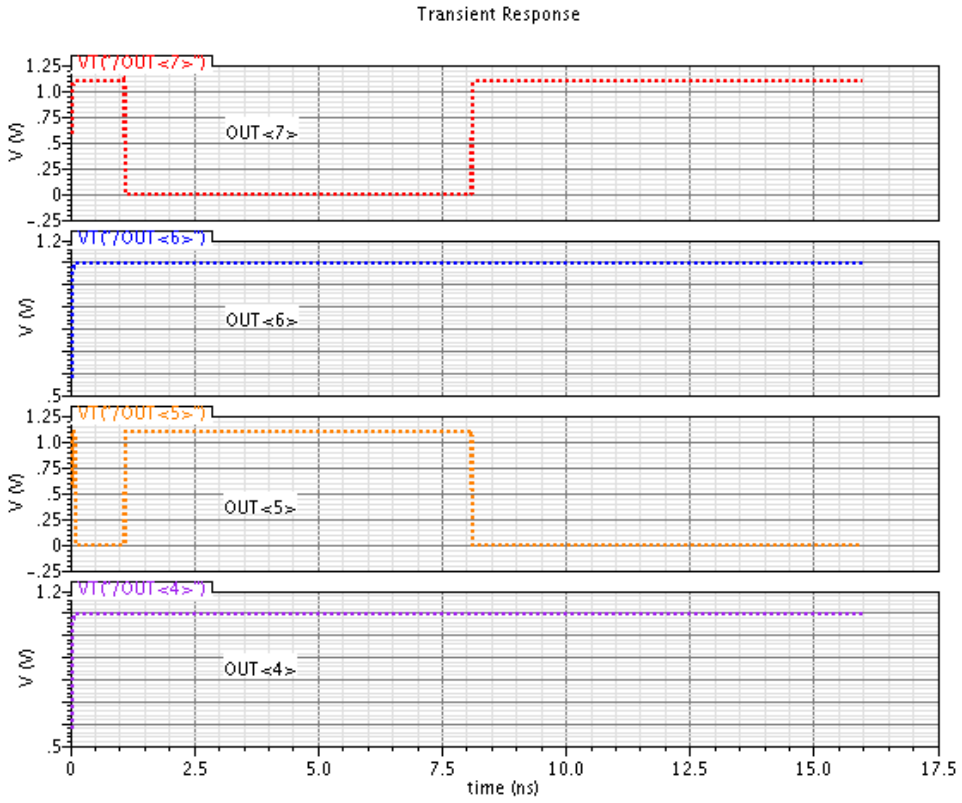


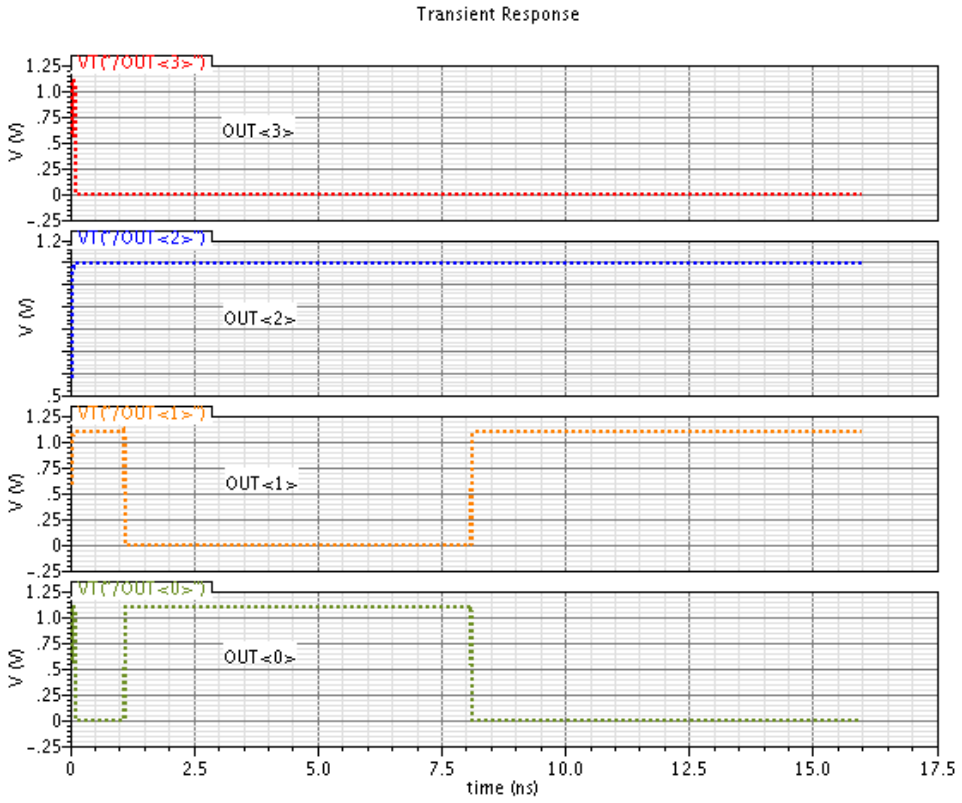
Graph of output when D is tied to high voltage: s2s1s0 = 011 yields high output as desired



REGISTER

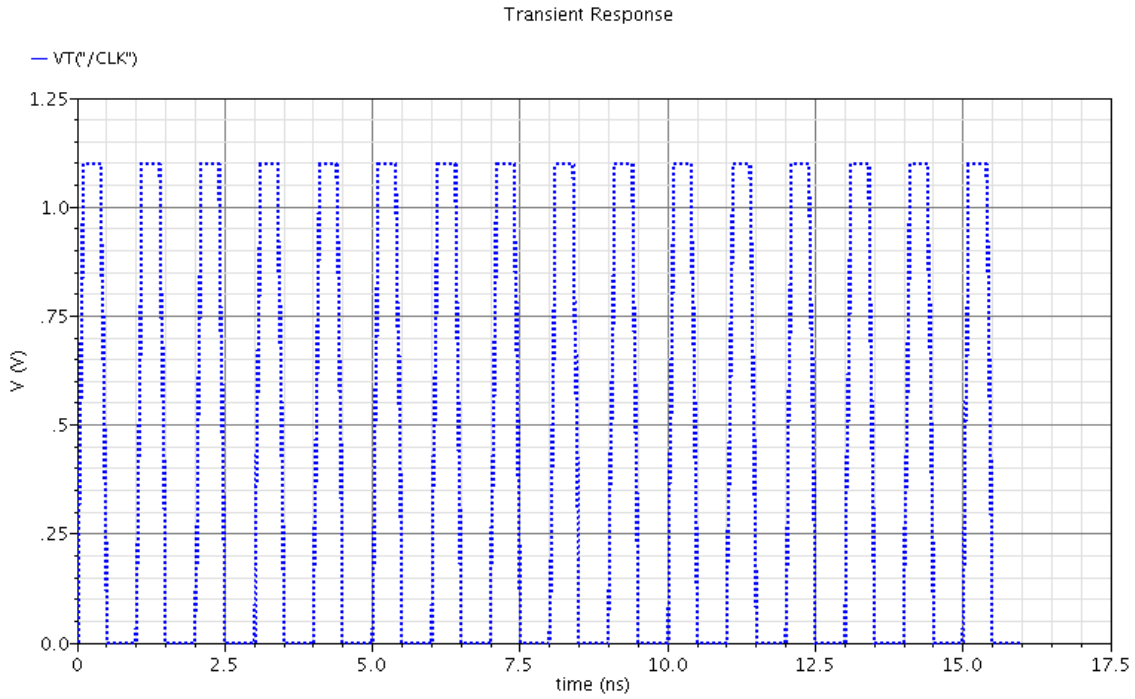
We then tested the register and were pleased to see the results supported our design. We decided to test a case that took into account all possible cases in a bit-by-bit respect. In other words, some bits changed from 0 to 1 or 1 to 0, and some remained static at 0 or 1. Therefore, we changed the input to the register from the binary value 01110101 t





The glitches at the beginning are simply discrepancies created when the input pulses were programmed prior to the first clock cycle. We see that these act as desired however. Bit 7

transitions from low to high as desired. Bit 6 remains static at high. Bit 5 transitions from high to low. Bit 4 remains constant at high. Bit 3 remains constant at low. Bit 2 remains constant at high. Bit 1 transitions from low to high. Lastly, Bit 0 transitions from high to low. These are all desired in a transition from the 01110101 to 11010110. Therefore, we conclude that our register works. Also, all of them changed at the same clock edge so we know that the edge-triggered design functions properly. The clock used is pictured below:



Conclusion: Final Schematic of DSP